Voisinage consistant pour le problème de satisfaisabilité

Lionel Paris, Djamal Habet, Belaïd Benhamou

Voisinage consistant pour le problème de satisfaisabilité

Lionel Paris[1] **Djamal Habet**[2] **Belaïd Benhamou**[1]

Laboratoire des Sciences de l'Information et de Systèmes (LSIS - UMR CNRS 6168)
[1]Université de Provence
[2]Université Paul Cézanne
Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
13397 MARSEILLE CEDEX 20
{lionel.paris, djamal.habet, belaid.benhamou}@lsis.org

Résumé

La plupart des méthodes de recherche locale pour le problème de satisfaisabilité traitent une interprétation complète et souvent inconsistante des variables du problème, et essaient de la réparer en changeant la valeur de vérité de certaines variables jusqu'à atteindre une solution. Nous proposons une nouvelle méthode de recherche locale qui gère des interprétations incomplètes, mais toujours consistantes, au lieu de complètes et inconsistantes. Cette méthode tente de prolonger l'interprétation partielle courante comme le ferait une méthode complète. Cependant, au lieu de déclencher un retour arrière (*backtrack*) lorsqu'un conflit survient, elle libère au moins une variable impliquée dans chaque clause falsifiée afin de restaurer la consistance. Ainsi, le voisinage exploré est toujours consistant alors que ce n'est pas le cas pour les algorithmes de recherche locale classiques. Notre méthode peut aussi tirer profit de certaines techniques efficaces issues des méthodes complètes comme la propagation unitaire et les heuristiques de choix de variables. Les résultats expérimentaux montrent la compétitivité de notre méthode par rapport à d'autres méthodes de recherche locale.

Abstract

Most of the local search methods for the satisfiability problem deal with a complete and inconsistent truth assignment of the problem variables, and try to repair it by switching the truth value of some variables until reaching a solution. We propose a new local search algorithm which works on consistent partial truth assignments, instead of inconsistent complete ones. This method attempts to extend a current partial consistent assignment as a complete method would do. However, instead of backtracking when a conflict arises, it frees at least one variable involved in each falsified clause to restore consistency. Thus, the explored neighborhood is always consistent whereas it is not the case for classical local search algorithms. Our method can also take advantages from some efficient techniques, such as unit propagation and heuristics of variable choice, wildly used in complete methods. Experimental results show the competitiveness of our method towards other local search methods.

1 Introduction

Le problème de satisfaisabilité de formules propositionnelles (SAT) qui consiste à décider si une formule booléenne mise sous forme normale conjonctive (CNF) est satisfaisable est l'un des problèmes NP-Complet les plus étudiés de part son importance pratique et théorique. Encouragé par une forte progression de la résolution pratique du problème SAT, un grand nombre d'applications, allant de la vérification formelle à la planification, sont codées et résolues en utilisant ce formalisme. Les premiers solveurs SAT, datant de 1962 [1], appartiennent à la classe des solveurs complets. La plupart de ces solveurs les plus efficaces sont basés sur un algorithme de recherche de type backtrack, appelé procédure de DPLL (Davis, Putnam, Logemann et

Loveland) [1]. Cet algorithme basique est couplé avec plusieurs méthodes d'apprentissage ou de filtrage importantes comme des formes étendues de propagation de contraintes booléennes, de pré-traitements, de détection des symétries, etc. L'impact de ces différentes améliorations dépend du type d'instances à résoudre. Par exemple, l'apprentissage est plus efficace pour résoudre des instances codant des problèmes réels que sur les problèmes aléatoires.

Cependant, pour faire face à la taille des instances SAT toujours plus importantes, la communauté a dû trouver un moyen de surmonter l'explosion combinatoire des méthodes complètes. Une réponse à ce problème fut l'introduction des méthodes de recherche locale, pour lesquelles l'espace de recherche est exploré de façon non systématique. Ces méthodes incomplètes s'avérèrent très efficaces sur beaucoup d'instances satisfaisables difficiles, aussi bien générées aléatoirement qu'issues d'applications réelles codées en SAT. Elles sont généralement basées sur une exploration partielle de l'espace de recherche [13, 6]. Plus précisément, elles partent d'une interprétation complète (de toutes les variables du problème) inconsistante, et tentent de la réparer en inversant la valeur de vérité de certaines variables jusqu'à ce que l'interprétation devienne un modèle (une solution de la formule CNF). Le choix de la variable dont on change la valeur peut être réalisé selon plusieurs heuristiques, généralement dépendantes du nombre de clauses satisfaites. Une des améliorations les plus importantes pour ces méthodes consiste à intégrer la stratégie de "marche aléatoire" pendant la résolution, produisant l'algorithme *Walksat* et ses variantes : Novelty, Novelty+, R-Novelty and R-Novelty+ [12, 9, 4, 5]. Ces méthodes contrôlent la marche aléatoire en modifiant une valeur p (paramètre de bruit), permettant d'introduire une phase de diversification à une phase d'amélioration stricte de la valeur de la fonction objectif (méthode de descente).

Les dernières méthodes de résolution du problème SAT offrent un large panel de méthodes de recherche locale. Par exemple, B. Mazure *et al.* ont introduit un algorithme basic de recherche taboue pour SAT, TSAT [8], où une longueur optimalede la liste taboue est mise en évidence pour les instances K-SAT aléatoires difficiles. Hirsch et Kojevnikov [3] on développé un solveur SAT efficace, UnitWalk, en combinant une recherche locale et un procédé d'élimination des clauses unitaires. F. Lardeux *et al.* [7] ont introduit une logique tri-valuée dans laquelle la valeur *indéterminée* est ajoutée. Ce formalisme a permis de concevoir un algorithme de recherche locale manipulant à la fois des interprétations partielles et complètes, et d'introduire de nouveaux mécanismes de diversification et d'intensification. Dans [14], K. Smyth *et al.* ont introduit un nouvel algorithme de recherche locale stochastique, Iterated Robust Tabu Search (IRoTS) pour le problème MAX-SAT (la version d'optimisation du problème SAT), qui combine deux méthodes de recherche locale , Iterated Local Search (ILS) [11] et Robust Tabu Search (RoTS) [15]. Enfin, dans [10], un moyen original d'exploiter la structure des problèmes SAT, généralement intégré au solveurs complets, a été introduit dans les solveurs incomplets. Cette nouvelle méthode s'avère très efficace sur les instances structurées. Tous ces algorithmes, excepté celui utilisant la logique trivaluée [7], autorisent la falsification de certaines clauses en visitant des configurations inconsistantes et complètes.

Dans cet article, nous proposons un nouvel algorithme de recherche locale pour SAT qui ne visite que des configurations consistantes. Au lieu de générer une interprétation complète et inconsistante, il essaie de construire un modèle en affectant les variables une à une à la manière d'une méthode complète. Chaque fois qu'un conflit survient, alors qu'une méthode complète aurait déclenché un retour-arrière (backtrack), notre méthode libère un ensemble minimum de variables impliquées dans les clauses falsifiées pour restaurer la consistance. Ainsi, pour une interprétation partielle donnée, ce mécanisme ne visite que le *voisinage consistant* de la configuration courante. Cette idée a été initiée dans le cadre des CPS binaires [16].

Le fait d'étendre l'interprétation comme le ferait une méthode complète nous permet d'utiliser des techniques efficaces issues des algorithmes complets telles que la propagation unitaire ou les heuristiques de choix de variables. Pour s'assurer que notre méthode n'instancie et ne libère pas toujours les même variables, nous couplons notre algorithme avec une liste taboue [2] pour laquelle nous verrons expérimentalement que la durée du statut tabou (*tabu tenure* en anglais) peut être empiriquement fixée. Ces expérimentations montreront aussi que cette méthode peut être efficace sur plusieurs classes d'instances, et résout même des instance que R-Novelty+ ne résout pas.

Le papier est organisé comme suit : tout d'abord, nous rappelons quelques définitions et introduisons les notations qui seront utilisées dans le reste de l'article. Ensuite nous décrivons la méthode du voisinage consistant pour SAT : $CN-SAT$. Des résultats expérimentaux sont ensuite présentés et discutés, avant de conclure et de présenter des pistes possibles à explorer pour améliorer le présent travail.

2 Préliminaires

2.1 Le problème SAT

Les formules du calcul propositionnel sont construites à l'aide des connecteurs logiques usuels ($\vee$, $\wedge$, $\neg$) et d'un ensemble $\mathcal{V}$ de variables booléennes (propositionnelles). Une *formule CNF* $\mathcal{F}$ est un ensemble (interprété comme une conjonction) de *clauses*, où chaque clause est un ensemble

(interprété comme une disjonction) de *littéraux*. Un littéral est une occurrence d'une variable propositionnelle ou de sa négation.

Une *interprétation* d'une formule booléenne est une affectation de ses variables à une valeur de vérité $\{vrai, faux\}$ (ou $\{0, 1\}$). Une variable x est satisfaite (resp. falsifiée) pour I si $I[v] = vrai$ (resp. $I[v] = faux$). Une interprétation I peut aussi être représentée par un ensemble de littéraux ne contenant pas un littéral et son opposé. Un littéral l appartient à I (resp. $\neg l$ appartient à I) si $I[l] = vrai$ (resp. $I[l] = faux$). Dans le cas d'une interprétation partielle (où certaines variables ne sont pas encore instanciées), les variables libres n'apparaissent dans l'interprétation ni sous forme positive, ni sous forme négative. On dit que ces variables ont une valeur de vérité *indéterminée* pour I et on note $I[v] = u$ (si x n'est pas instanciée).

On définit $\mathcal{F}_I$ comme la formule simplifiée par l'interprétation partielle I, éventuellement réduite à $Vrai$ ou $Faux$. Formellement, $\mathcal{F}_I = \{C \backslash l | C \in \mathcal{F}, I \cap C = \emptyset, \neg l \in I\}$. Un *modèle* pour une formule est une interprétation qui satisfait la formule. Le problème SAT est le problème de décision de la satisfaisabilité d'une CNF (existence d'un modèle).

2.2 Consistance locale

Une interprétation partielle d'un ensemble de variables $\mathcal{S} \subset \mathcal{V}$ est localement consistante si et seulement si elle satisfait toutes les clauses n'impliquant que les variables de $\mathcal{S}$. De plus, une interprétation partielle qui ne peut pas appartenir à un modèle est un *nogood*.

2.3 Recherche taboue

Une recherche taboue (RT) est une méta-heuristique conçue pour aborder les problèmes d'optimisation combinatoires [2]. Contrairement aux approches purement aléatoires, RT est basée sur la croyance qu'une recherche intelligente devrait inclure une forme plus systématique d'exploration, basée sur une mémoire et un apprentissage adaptatifs. RT peut être décrite comme une forme de recherche dans un voisinage avec un ensemble de composants critiques et complémentaires.

Pour un problème d'optimisation (S, f), caractérisé par un espace de recherche S et une fonction objectif f, un voisinage $\mathcal{N}$ est introduit, qui associe à chaque configuration s de S un sous-ensemble non vide $\mathcal{N}(s)$ de S. Un algorithme RT typique commence avec une configuration initiale s dans S, puis visite successivement une série de "meilleures" configurations locales en suivant la fonction de voisinage. À chaque itération, un des meilleurs voisins $s' \in S$ est sélectionné pour devenir la configuration courante, même si s' n'améliore pas la configuration courante du point de vue de la fonction objectif.

Pour éviter l'apparition de cycle lors du parcours de l'espace de recherche, et permettre à la recherche de quitter des optimums locaux, une liste taboue est introduite. Ceci ajoute une composante mémoire à la méthode. En effet, une liste taboue maintient un historique sélectionné H, composé des configurations visitées précédemment, ou plus généralement, certains attributs pertinents de ces configurations. Une stratégie simple consiste à empêcher les configurations contenues dans H d'être considérées à nouveau pour les k prochaines itérations (k est appelé la *durée taboue*). Cette durée peut varier en fonction des différents critères et dépend généralement du problème considéré. À chaque itération, RT cherche le meilleur voisin de s dans $\mathcal{N}(H, s)$ maintenu à jour dynamiquement, au lieu de $\mathcal{N}(s)$ lui-même.

Quand les attributs des configurations sont enregistrés dans la liste taboue au lieu des configurations elles-mêmes, la liste taboue peut empêcher certaines configurations non visitées, mais néanmoins intéressantes, d'être considérées. Des critères d'aspiration peuvent être utilisés pour palier ce problème. Le critère d'aspiration le plus utilisé consiste à annuler le statut tabou d'un mouvement qui aboutit à une configuration meilleure que toutes celles obtenues jusqu'à lors.

3 Composants de CN-SAT

Nous allons décrire dans cette section la méthode que nous proposons (CN-SAT) pas à pas. Nous commençons par les définitions de configurations partielles et de la notion de voisinage consistant, puis nous détaillons les différents composants de l'algorithme que nous proposons.

3.1 Configurations partielles

Dans la plupart des algorithmes de recherche locale (RL) pour SAT, une configuration est une interprétation de toutes les variables du problème. L'espace de recherche correspond à l'ensemble de toutes les configurations qui sont soit des modèles, soit des interprétations qui mènent à une contradiction. Dans le cas d'une instance SAT à n variables, une configuration peut être représentée par un vecteur de dimension n ; $s = (v_1, v_2, ..., v_n)$ tel que $\forall i = 1 \cdots n$, $v_i \in \{0, 1\}$ est la valeur de vérité de la variable x_i.

Comme nous l'avons annoncé dans la section 1, dans notre approche l'espace de recherche est constitué de *configurations partielles* notées $s_i = (v_{i_1}, v_{i_2}, ..., v_{i_{n_i}})$ où n_i variables sont instanciées ($n_i = |s_i| \leq n$). Dans ces inter-

prétations, les variables non encore instanciées sont affectées à la valeur *indéterminée* u.

3.2 Voisinage consistant

Les algorithmes de recherche locale classiques remplacent une configuration s par une nouvelle configuration obtenue en opérant un mouvement, noté $mv(x_i,\ v_i)$, qui modifie la valeur courante de la variable x_i de $s[x_i]$ à v_i. Dans notre approche, nous définissions un mouvement comme suit : tout d'abord, les mouvements ne s'appliquent qu'aux variables libres (instanciée à la valeur u). Ensuite, ils sont suivis par une restauration de la consistance qui libère au moins une variable par clause falsifiée. L'ensemble des configurations ainsi atteintes représente le voisinage consistant de s.

De cette manière, seules les configurations localement consistantes sont évaluées dans $\mathcal{N}$, configurations qui peuvent être distantes les unes des autres par plus d'une affectation. Pour rester concis, nous conserverons la notation $mv(x_i,\ v_i)$ pour la nouvelle définition d'un mouvement, mais un tel mouvement consiste en un ancien mouvement, dit élémentaire, $mv(x_i, v_i)$ suivi éventuellement d'un certain nombre de mouvements de la forme $mv(x_j, u)$ nécessaires à la restauration de la consistance.

3.3 Évaluation du voisinage

Les méthodes de recherche locale comme Walksat partent d'une configuration générée aléatoirement, qui est complète et inconsistante, et tentent de la réparer jusqu'à satisfaire toutes les clauses. En conséquence, la fonction objectif est celle qui maximise le nombre de clauses satisfaites, et le passage d'une configuration à une autre est guidé par la valeur de cette fonction.

Dans l'approche $CN\text{-}SAT$, le critère d'évaluation d'une configuration est *le nombre de variables instanciées* dans s. Ainsi, la fonction objectif à maximiser est $f = |s|$. Cependant, plutôt que d'évaluer un mouvement en comptant le nombre de variables instanciées après l'avoir effectué, nous calculons le nombre de variables libérées si ce mouvement est effectué. En effet, la difficulté réside dans le fait de libérer le moins de variables possible à chaque mouvement, ainsi, une fois les variables à libérer connues, avec leur nombre, nous pouvons immédiatement savoir combien de variables seront instanciées au total après ce mouvement. Pour ce faire, une fonction δ est calculée.

Plus précisément, si s' est la configuration obtenue à partir de s en appliquant le mouvement $mv(x_i, v_i)$ alors $\delta(i, k)$ retourne le nombre de variables déjà instanciées qui seront libérées (*i.e.* affectées à u) si x_i est affectée à $v_i = k$, $k \in \{0,\ 1\}$. La fonction δ est calculée en utilisant l'algorithme 1, où δ_{min} (ligne 6) est la valeur minimale de δ pour l'ensemble du voisinage de s alors que $\delta_{min}^{\neg tabu}$ (ligne 10) est la valeur minimale δ_{min} biaisée par le statut tabou de certaines variables *i.e.* seuls les mouvements non tabous sont considérés.

Algorithm 1 $evaluate\_\mathcal{N}(s)$

Fonction $evaluate\_\mathcal{N}(s)$

1: $\delta_{min} \leftarrow n$;
2: $\delta_{min}^{\neg tabu} \leftarrow n$;
3: $\mathcal{L}_{cand} \leftarrow \emptyset$;
4: **pour** *tout* $x_i \in s$, *tel que* $x_i = u$ **faire**
5: **pour** *chaque* $k \in \{0,\ 1\}$ **faire**
6: **si** $\delta(i,k) < \delta_{min}$ **alors**
7: $\delta_{min} \leftarrow \delta(x_i, k)$;
8: **fin si**
9: **si** (x_i, k) n'est pas tabou **alors**
10: **si** $\delta(x_i, k) < \delta_{min}^{\neg tabu}$ **alors**
11: $\delta_{min}^{\neg tabu} \leftarrow \delta(x_i, k)$;
12: $\mathcal{L}_{cand} \leftarrow (x_i, k)$;
13: **sinon**
14: **si** $\delta(x_i, k) = \delta_{min}^{\neg tabu}$ **alors**
15: $\mathcal{L}_{cand} \leftarrow \mathcal{L}_{cand} \cup \{(x_i, k)\}$;
16: **fin si**
17: **fin si**
18: **fin si**
19: **fin pour**
20: **fin pour**
21: **retourner** $\delta_{min}, \mathcal{L}_{cand}$

3.4 Calcul de δ

3.4.1 Exemple

Rappelons que la valeur δ correspond au nombre de variables qui doivent être libérées (affectées à u) pour un mouvement donné.

Par exemple, considérons l'ensemble des clauses suivant :

$(c_1) : \neg x_1 \vee x_2 \vee \neg x_4 \vee \neg x_6$ $(c_2) : \neg x_1 \vee \neg x_5$
$(c_3) : x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_7$ $(c_4) : x_2 \vee \neg x_3 \vee \neg x_5$
$(c_5) : \neg x_6 \vee \neg x_7 \vee x_4$ $(c_6) : \neg x_8 \vee x_4$
$(c_7) : \neg x_6 \vee \neg x_7 \vee x_5$ $(c_8) : \neg x_8 \vee x_5$

et la configuration s suivante : $s = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (1, 0, 1, u, u, 1, 1, 1)$, où les variables x_4 et x_5 sont libres.

Avant d'évaluer les différents mouvements, donnons la table de vérité du $\vee$ pour la logique tri-valuée $\{0, 1, u\}$:

Au regard de ce tableau, toutes les clauses de notre exemple prennent la valeur u pour la configuration s, nous

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1
0	u	u
u	0	u
1	u	1
u	1	1
u	u	u

TAB. 1 – Table de vérité du $\vee$ pour la logique tri-valuée.

devons donc calculer δ pour les affectations des variables x_4 et x_5 à 1 et 0 :

- $\delta(x_4, 1) = 1$ cela signifie que si l'on affectait la valeur $vrai$ à x_4 nous devrions libérer *une* variable pour restaurer la consistance : ici x_2. En fait, affecter x_4 à $vrai$ falsifierait les clauses (c_1) et (c_3). La libération de x_2 supprime cette falsification.
- $\delta(x_4, 0) = 2$: si x_4 était affectée à $faux$ nous devrions libérer *deux* variables : x_6 et x_8 sinon les clauses (c_5) et (c_6) seraient falsifiées.
- $\delta(x_5, 1) = 2$, x_1 et x_2 devraient être libérées sinon les clauses (c_2) et (c_4) seraient falsifiées.
- $\delta(x_5, 0) = 2$, x_6 et x_8 devraient être libérées pour éviter de falsifier les clauses (c_7) et (c_8).

Après cette évaluation, le mouvement sélectionné pour l'itération courante devrait minimiser la valeur de δ. Ainsi, $mv(x_4, 1)$ serait sélectionné (où x_4 est affecté à $vrai$ et x_2 à u). Notons que cet exemple ne prend pas en compte le statut tabou des différents mouvements.

3.4.2 Algorithme d'évaluation approchée de δ

Calculer l'ensemble minimal des variables à libérer après une affectation est équivalent au problème du calcul d'un transversal minimal (minimal Hitting-Set) d'un hypergraphe, qui est NP-Difficile.

Reprenons l'exemple précédent, affecter x_4 à $vrai$ viole les clauses (c_1) et (c_3) et satisfait (c_5) et (c_6). Pour maintenir la consistance, nous devons sélectionner un nombre de variables apparaissant dans (c_1) et (c_3) (excepté x_4 pour ne pas boucler).

Si on réduit (c_1) et (c_3) en supprimant x_4, on obtient les sous-clauses $(c'_1) : \neg x_1 \vee x_2 \vee \neg x_6$ et $(c'_3) : x_2 \vee \neg x_3 \vee \neg x_7$ qui sont falsifiées. On peut les représenter par un hypergraphe $G = (V, E)$ où $V = \{x_1, x_2, x_3, x_6, x_7\}$ est l'ensemble des sommets et $E = \{(x_1, x_2, x_6), (x_2, x_3, x_7)\}$ est l'ensemble des hyper-arêtes *i.e.* chaque hyper-arête représente une clause. Le transversal minimal de l'hypergraphe G est l'ensemble $\{x_2\}$. Pour approximer cet ensemble minimal, nous utilisons un algorithme glouton simple basé sur le degré des sommets. Cet algorithme choisit successivement le sommet ayant le degré le plus fort dans les hyper-arêtes non traitées, puis le retire et marque les hyper-arêtes le contenant comme étant traitées. Ce procédé est répété jusqu'à ce que toutes les hyper-arêtes soient diminuées d'au moins un sommet. L'ensemble des variables associées aux sommets qui ont été retirés constitue notre ensemble minimal de variables à libérer.

3.4.3 Algorithme d'évaluation exacte de δ

Parallèlement à l'algorithme glouton, nous avons développé un algorithme de calcul exact pour δ décrit par l'algorithme 2. Il commence par initialiser un certain nombre de structures (lignes 1 à 9), puis traite les clauses indépendantes[1] en choisissant arbitrairement, pour chacune d'elles, une de leur variable à libérer (fonction $choose\_var\_in\_clause(c)$). Ensuite un algorithme simple de Branch & Bound est appelé (cf. algorithme 3) pour finir de produire l'ensemble minimal de variables à libérer (δ). Pendant sa construction, cette ensemble est appelé HS dans les algorithmes 2 et 3.

3.5 Gestion de la liste taboue

Nous avons décrit une méthode utilisant des mouvements multi-attributs (impliquant plusieurs variables), pouvant ainsi produire des cycles dans l'espace de recherche. Pour éviter de tels phénomènes nous introduisons une liste taboue de mouvements interdits. Avant de détailler les spécificités de cette liste, rappelons d'abord qu'un mouvement $mv(x_i, v_i)$ consiste en l'affectation de v_i à x_i, puis la libération de r variables $x_{j_1}, ..., x_{j_r}$ pour pallier la falsification de certaines clauses. Si un tel mouvement est appliqué alors les mouvements $mv(x_{j_1}, v_{j_1}), ..., mv(x_{j_r}, v_{j_r})$ seront considérés tabous durant un certain nombre d'itérations pour empêcher de libérer x_i trop tôt.

Par ce procédé, nous rendons taboues toutes les valeurs de variables libérées qui sont en conflit avec la nouvelle valeur de x_i. Nous définissons dynamiquement la durée de l'interdiction d'une affectation de la variable x_k à la valeur v_k en fonction du nombre de fois où x_k a été affectée à v_k depuis le début de la recherche. Ce nombre que nous notons $freq(x_k, v_k)$ correspond à la fréquence du mouvement $mv(x_k, v_k)$. À chaque itération $iter$, nous fixons le statut tabou pour chaque couple (variable,valeur) des variables libérées à : $tabou(x_j, k) = iter + \alpha \times freq(x_j, k)$, où k est la valeur à laquelle la variable x_j était affectée avant

[1] Nous considérons indépendantes les clauses dont les littéraux qui les composent n'apparaissent dans aucune autre clause.

Algorithm 2 Exact Min Hitting-Set

Fonction ExactMinHittingSet

Entrée : x_i : une variable, k : une valeur booléenne, $\mathcal{F}$: une formule CNF, s : une interprétation partielle (modèle en cours de construction)

Sortie : un ensemble de taille minimale de variables à libérer pour restaurer la satisfaisabilité de $\mathcal{F}$, si on étend le modèle partiel s avec $x_i \leftarrow k$.

1: $SC \leftarrow$ toutes les clauses falsifiées par l'affectation de x_i à k ;
2: retirer x_i (ou $\neg x_i$ en fonction de la valeur de k) de chaque clauses de SC ;
3: $SL \leftarrow \emptyset$; {un ensemble vide de littéraux}
4: $MinCover \leftarrow \emptyset$; {le futur transversal minimal}
5: $B \leftarrow |SC| + 1$; {la taille du transversal minimal dans le pire des cas}
6: entier $fixed[|SC|]$; {un tableau de $|SC|$ entiers}
7: **pour** chaque clause $c \in SC$ **faire**
8: $\quad fixed[c] \leftarrow B$;
9: **fin pour**
10: **pour** chaque clause $c \in SC$ telle que pour chaque clause $c' \in SC$ ($c \neq c'$), $c \cap c' = \emptyset$ **faire**
11: $\quad SL \leftarrow SL \cup choose\_var\_in\_clause(c)$; {Sélection de la variable à libérer pour éviter l'inconsistance}
12: $\quad fixed[c] \leftarrow -1$; {la clause c n'est plus falsifiée}
13: **fin pour**
14: $c \leftarrow$ première clause de SC telle que $fixed[c] \neq -1$;
15: **pour** chaque littéral l de c **faire**
16: $\quad$ MinHittingSet($SL \cup l, 0, B, l, SC, fixed, HS$) ;
17: **fin pour**
18: $MinCover \leftarrow HS$;
19: **retourner** $MinCover$

Algorithm 3 Min Hitting-Set

Procédure MinHittingSet

Entrée : SL : un ensemble de littéraux, $depth$, B : entier, lit : un littéral, SC : un ensemble de clauses, $fixed[|SC|]$: entier, HS : un ensemble de littéraux.

Sortie : HS : un des plus petits transversaux minimaux de SC, et B : sa taille.

1: **si** $|SL| <$ B **alors**
2: $\quad$ **pour** chaque clause c de CS telle que $lit \in c$ et $fixed[c] \geq depth$ **faire**
3: $\quad\quad fixed[c] \leftarrow depth$;
4: $\quad$ **fin pour**
5: $\quad$ **si** il n'y a plus de clause falsifiée **alors**
6: $\quad\quad B \leftarrow |SL|$;
7: $\quad\quad HS \leftarrow SL$;
8: $\quad$ **sinon**
9: $\quad\quad c \leftarrow$ première clause de SC telle que $fixed[c] \geq depth$;
10: $\quad\quad$ **pour** chaque littéral l de c **faire**
11: $\quad\quad\quad$ HittingSet($SL \cup l$, $depth + 1$, B, l, $SC, fixed$, HS) ;
12: $\quad\quad$ **fin pour**
13: $\quad$ **fin si**
14: **fin si**

de la libérer et α est un coefficient utilisé pour paramétrer la longueur de la liste taboue. Ainsi, le statut tabou du couple (x_j, v_j) est proportionnel à la fréquence à laquelle x_j a été affectée à v_j depuis le début de la recherche et l'empêche donc d'y être affecté trop souvent. Par ailleurs, le statut tabou d'un mouvement $mv(x_i, v_i)$ tel que s' = s+$mv(x_i, v_i)$ est annulé si $|s'| > |s^*|$ (où s^* est la configuration la plus complète obtenue depuis le début de la recherche). Ceci correspond au critère *d'aspiration*. Dans la fonction $evaluate\_\mathcal{N}$, la condition *is not tabu* correspond à $(tabu(x_i, k) \leq iter \vee |s'| > |s^*|)$.

3.6 Diversification et intensification

Le but de la diversification est d'aider la recherche à quitter les minima locaux de la fonction objectif. Dans ce but, nous introduisons un procédé de diversification simple, mais souvent efficace, qui consiste à redémarrer la recherche avec une nouvelle configuration partielle et consistante différente de la précédente. Cette nouvelle tentative s'accompagne toujours d'une initialisation de la liste taboue à vide. Plus précisément, nous autorisons un nombre fixé de tentatives, et à chacune d'elles, la recherche taboue explore l'espace de recherche d'un point initial (configuration) différent des précédents essais.

Par opposition à la diversification, l'intensification tente de concentrer la recherche dans des zones prometteuses de l'espace de recherche. Pour atteindre ce but, nous vidons la liste taboue, augmentons le coefficient α (pour augmenter la durée de restriction des mouvements tabous), et nous relançons la recherche à partir de la meilleur solution trouvée s^*. La phase d'intensification est appelée sous deux conditions : tous les mouvements candidats sont tabous ($\mathcal{L}_{cand} = \emptyset$), ou bien après qu'un certain nombre d'itérations préalablement fixé soit atteint.

3.7 Voisinage consistant pour SAT

La combinaison de tous les points, décrits dans les précédentes sections, nous amène à un algorithme de recherche taboue sur un voisinage consistant pour SAT ($CN\text{-}SAT$) :

L'algorithme $greedy()$ tente d'affecter une variable libre tant qu'aucune clause n'est falsifiée. Sa caractéristique principale est de produire une configuration partielle dif-

Algorithm 4 $CN\text{-}SAT$

Procedure $CN\text{-}SAT$

pour try = 1 **to** max_tries **faire**
  $s \leftarrow greedy()$; $s^* \leftarrow s$; $iter \leftarrow 0$;
  pour move =1 **to** max_moves **faire**
    répéter
      $(\delta_{min}, \mathcal{L}_{cand}) \leftarrow evaluate\_\mathcal{N}(s)$;
      si $\mathcal{L}_{cand} \neq \emptyset$ **alors**
        $iter++$;
        $(x_i, k) \leftarrow Select(\mathcal{L}_{cand})$;
        $propagate\_move(x_i, v_i)$;
        $freq(x_i, v_i)++$;
        si $|s| > |s^*|$ **alors**
          $s^* \leftarrow s$;
          si $|s^*| = n$ **alors**
            retourner s^*
          fin si
        fin si
      fin si
    jusqu'à ce que $\mathcal{L}_{cand} = \emptyset$
    $Intensify(s^*)$;
  fin pour
fin pour
retourner s^*

férente à chaque appel. Cela permet un certaine diversification de la recherche comme expliqué précédemment. À chaque tentative, nous générons ainsi la première configuration partielle consistante, et effectuons un certain nombre de mouvements jusqu'à ce qu'une interprétation complète consistante soit trouvée, ou bien jusqu'à atteindre un nombre fixé de mouvements, ou encore jusqu'à ce qu'aucun mouvement ne soit candidat. Dans ce dernier cas, la phase d'intensification ($Intensify(s^*)$) est lancée. La fonction $Select(\mathcal{L}_{cand})$ choisit aléatoirement un mouvement candidat dans $\mathcal{L}_{cand}$ comme décrit dans les sections 3.3 et 3.4. La fonction $propagate\_move(x_i, v_i)$ affecte x_i à v_i et libère les variables pour réparer les conflits éventuels, tout en réalisant la propagation unitaire.

4 Expérimentation

Pour évaluer les performances de $CN\text{-}SAT$, nous avons réalisé des expérimentations sur des instances 3-SAT aléatoires difficiles[2] et sur des instances structurées. Nous le comparons avec l'algorithme de recherche locale R-Novelty+, considéré comme l'un des solveurs incomplets les plus efficaces pour SAT. $CN\text{-}SAT$ est écrit en C, compilé sous linux et testé sur un ordinateur P_{IV} cadencé à 3.0 Ghz avec 1 Go de RAM. Les paramètres utilisés pour $CN\text{-}SAT$ et R-Novelty+ sont : *max_mouvements* = 10^6 par tentative et le temps limite d'exécution est fixé à 300 secondes pour les deux méthodes. Le paramètre de bruit de R-Novelty+ est fixé à sa valeur par défaut. Le paramètre α utilisé dans le calcul de la durée taboue d'un mouvement est fixé à 1.9 [3] pendant la phase de RT, puis à $2,8$ durant les phases d'intensification. Nous avons codé les deux méthodes de calcul de δ (approchée et exacte), ce qui nous amène à $CN\text{-}SAT$ Approx et $CN\text{-}SAT$ Exact dans les tableaux suivants.

4.1 Instances structurées

Nous avons choisit des instances SAT structurées et satisfaisables disponibles sur le site `http://www.satlib.org`. Dans le tableau 2, les caractères "- -" signifient que l'instance n'a pas été résolue.

Instances	R-Novelty+		$CN\text{-}SAT$ Approx		$CN\text{-}SAT$ Exact	
	Temps	# Flips	Temps	# Flips	Temps	# Flips
qg1-07	15,21	1580369	13,63	112616	0,85	4432
qg1-08	- -	- -	- -	- -	144,11	310072
qg2-07	18,61	2007560	184,54	1281703	45,08	277824
qg2-08	- -	- -	- -	- -	18,8	35191
qg3-08	60,67	35467402	- -	- -	29,75	1107726
qg4-09	46,8	24648700	61,33	1989338	41,83	1151820
qg5-11	- -	- -	- -	- -	17,3	32108
qg6-09	- -	- -	129,09	1089167	121,19	1073697
qg7-09	- -	- -	0,29	1659	6,61	40430
qg7-13	- -	- -	- -	- -	- -	- -
bw_large.a	0	1688	0,07	5575	0,18	19046
bw_large.b	0,14	137167	20,22	1248845	1,61	90192
bw_large.c	66,57	36721914	- -	- -	- -	- -
bw_large.d	- -	- -	- -	- -	- -	- -
ais10	3,22	1960070	- -	- -	12,84	1849783
ais12	44,98	21858330	- -	- -	- -	- -
ais6	0,03	37574	10,44	5000029	0	1794
ais8	0,39	342178	44,12	15000127	1,3	288649
par8-1-c	0	4245	0,06	17641	0,02	4962
par8-1	5,07	11513624	0,01	2648	0,01	1152
par8-2-c	0,12	282750	0,02	6792	0,01	3602
par8-2	4,09	9308173	36,56	8001315	10,16	2003006
par8-3-c	5	11786722	48,27	13000763	3,41	789625
par8-3	7,25	16001600	36,6	8001176	6,43	1001334
par8-4-c	0,4	905109	3,55	1000756	0,53	133122
par8-4	54,92	122546333	0,01	1573	0,01	1547
par8-5-c	1,22	3000128	4,49	1005056	2,86	567755
par8-5	22,07	49074973	237,6	43377770	30,6	5359820
par16-1-c	- -	- -	- -	- -	199,45	13246954
Par16-2-c	- -	- -	- -	- -	- -	- -
Par16-3-c	- -	- -	- -	- -	- -	- -
par16-4-c	- -	- -	- -	- -	58,06	3917540
par16-5-c	- -	- -	- -	- -	149,99	8495650

TAB. 2 – $CN\text{-}SAT$ vs. R-Novelty+ sur des instances structurées.

Sur les instances *latin square* (notées qg*), $CN\text{-}SAT$ Exact est plus efficace que R-Novelty+ et $CN\text{-}SAT$ Approx. Sur les instances de planification (notées bw_large*), R-Novelty+ est capable de résoudre une instance de plus,

[2] Difficiles dans le sens où le ratio $\frac{Nb\ clauses}{Nb\ variables}$ est au seuil de 4.25

[3] Cette valeur a été choisie empiriquement car c'est celle qui a fournie les meilleurs résultats.

Size		R-Novelty+			CN-SAT Approx			CN-SAT Exact		
# V	# C	% Résolu	Temps	# Flips	% Résolu	Temps	# Flips	% Résolu	Temps	# Flips
100	425	100,00%	0,02	46198,1	100%	0,32	237870	100%	0,07	29685,26
150	637	100,00%	0,08	157475,51	100%	2,09	1261123	100%	0,93	298455,54
200	850	100,00%	0,04	64346,41	100%	4,62	1335377	100%	3,02	809924,87
250	1062	100,00%	1,22	2062181,27	100%	11,92	4703387	100%	20,98	5006912,67
300	1275	100,00%	0,68	1154850,27	100%	21,36	6670677	93%	33,67	7140076,12
350	1487	100,00%	0,87	1500417,86	97%	39,85	11338048	85,00%	50,99	9920538,94
400	1700	100,00%	7,84	13106152,22	87%	41,02	9980921	68,00%	70,96	12686260,23
450	1912	100,00%	3,39	5177363,17	85%	52,47	11519389	45,00%	133,04	22190221,05

TAB. 3 – CN-SAT vs. R-Novelty+ sur des instances 3-SAT aléatoire (seuil 4,25)

mais les performances des trois algorithmes sont comparables. Sur les instances des séries "All intervall" (notées ais*), R-Novelty+ semble être meilleur. Finalement, sur les instances parity (notées parity*), on peut remarquer que sur les petites instances (parity8*), CN-SAT Exact semble le plus efficace, ce qui se confirme sur les instances de plus grande taille (parity16*), car seul la version CN-SAT Exact arrive à en résoudre certaines. Pour conclure, on peut dire que CN-SAT Exact est plus performant que CN-SAT Approx sur les instances structurées et que sur certaines de ces instances, CN-SAT Exact domine aussi R-Novelty+.

4.2 Instances 3-SAT aléatoires

Nous avons généré aléatoirement 8 classes d'instances aléatoires et chacune d'elles est composée de 50 instances difficiles et satisfaisables. Le nombre des variables (qui définit les 8 classes) varie de 100 à 450 avec un pas de 50.

Dans le tableau 3, les colonnes "% solved", "Time" et "#Flips" correspondent à la moyenne du nombre d'instances résolues, la moyenne du temps requis (en seconde) pour la résolution et le nombre total de mouvements réalisés. Pour les instances à 100, 150, 200, 250 et 300 variables, CN-SAT* et R-Novelty+ résolvent sensiblement la même quantité d'instances, même si R-Novelty+ est plus rapide. Pour le reste des instances, R-Novelty+ dépasse les deux version de CN-SAT aussi bien en terme du temps de calcul que du pourcentage d'instance résolues. On peut également noté que CN-SAT Approx est plus efficace que CN-SAT Exact sur les plus grandes instances aléatoires.

5 Conclusions et perspectives

Nous avons proposé une nouvelle méthode de recherche locale pour résoudre le problème SAT. Sa principale caractéristique est que son espace de recherche est composé de configurations partielles consistantes. À chaque tentative, une configuration partielle différente est générée, s'en suit une construction de voisinage assurant la satisfaisabilité de toutes les clauses contenant les variables déjà instanciées. Ainsi, au lieu de réparer une configuration courante, comme dans les méthodes de recherche locale classiques, nous essayons à chaque étape d'instancier une variable tout en préservant la consistance. De plus, nous avons proposé deux algorithmes, un exact et l'autre approché, pour évaluer le voisinage, donnant lieu à deux versions de la méthode de voisinage consistant que nous avons expérimentées.

Les expérimentations présentées sont prometteuses, même si certaines améliorations devraient être effectuées. En fait, sur les instances difficiles testées, il ne reste vraiment que peu de variables que notre algorithme ne parvient pas à affecter. Cette observation peut être expliquée par la présence de nogoods qui empêchent d'atteindre une interprétation complète. Nous envisageons donc notamment de détecter ces nogoods pour introduire un autre mécanisme de diversification.

Cette approche peut également être adaptée pour traiter la variante MAX-SAT du problème SAT en changeant la fonction objectif pour maximiser le nombre de clauses satisfaites au lieu du nombre de variables instanciées. Enfin nous pensons utiliser cette idée du voisinage consistant dans les méthodes évolutives du type *Scatter Search, algorithmes génétiques* et la *mimétique* qui manipulent des populations d'individus.

Références

[1] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[2] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[3] Edward A. Hirsch and Arist Kojevnikov. Unitwalk : A new sat solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1-4) :91–111, 2005.

[4] Holger H. Hoos. The Run-Time Behaviour of Stochastic Local Search Algorithms for SAT. In *In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 661–666, 1999.

[5] Holger H. Hoos and Thomas Stützle. Local search algorithms for sat : An empirical evaluation. *J. Autom. Reason.*, 24(4) :421–481, 2000.

[6] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search : Foundations and applications*. Elsevier / Morgan Kaufmann, San Francisco, CA, 2004.

[7] Frédéric Lardeux, Frédéric Saubion, and Jin-Kao Hao. Three Truth Values for the SAT and MAX-SAT Problems. In *Proc. of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Lecture Notes in Computer Science. Springer, aug 2005.

[8] Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. Tabu search for SAT. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 281–285. AAAI Press, July 27–31 1997.

[9] David McAllester, Bart Selman, and Henry Kautz. Evidence for Invariants in Local Search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97*, pages 321–326, Providence, Rhode Island, 1997. MIT Press.

[10] Duc Nghia Pham, John Thornton, and Abdul Sattar. Building Structure into Local Search for SAT. In *Proceedings of IJCAI'07*, pages 2359–2364, Hyderabad, India, January 2007.

[11] Helena Ramalhinho-Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated Local Search. *International Series in Operations Research & Management Science*, 57 :321–353, 2000.

[12] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise Strategies for Improving Local Search. In MIT press, editor, *Proceedings of the 12th National Conference on Artificial Intelligence AAAI'94*, volume 1, pages 337–343, 1994.

[13] Bart Selman, Hector J. Levesque, and David Mitchell. A New Method for Solving Hard Satisfiability Problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI'92*, pages 440–446, Menlo Park, California, 1992.

[14] Kevin Smyth, Holger H. Hoos, and Thomas Stützle. Iterated Robust Tabu Search for MAX-SAT. In *Canadian Conference on AI*, pages 129–144, 2003.

[15] Éric Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17(4-5) :443–455, 1991.

[16] Michel Vasquez, Audrey Dupont, and Djamal Habet. Consistent Neighborhood in a Tabu Search. In *Metaheuristics : Progress as real Problem Sovers, MIC 2003 Post-conference volume*. Kluwer Academic Publishers, 2005.